Hardware Accelerator for the Tate Pairing in Characteristic Three Based on Karatsuba-Ofman Multipliers

Jean-Luc Beuchat[1], Jérémie Detrey[2], Nicolas Estibals[2], Eiji Okamoto[1], and Francisco Rodríguez-Henríquez[3]

[1] Graduate School of Systems and Information Engineering, University of Tsukuba, 1-1-1 Tennodai, Tsukuba, Ibaraki, 305-8573, Japan
[2] CACAO project-team, LORIA, INRIA Nancy - Grand Est, Bâtiment A, 615, rue du Jardin Botanique, 54602 Villers-les-Nancy Cédex, France
[3] Computer Science Department, Centro de Investigación y de Estudios Avanzados del IPN, Av. Instituto Politécnico Nacional No. 2508, 07300 México City, México

Abstract. This paper is devoted to the design of fast parallel accelerators for the cryptographic Tate pairing in characteristic three over supersingular elliptic curves. We propose here a novel hardware implementation of Miller's loop based on a pipelined Karatsuba-Ofman multiplier. Thanks to a careful selection of algorithms for computing the tower field arithmetic associated to the Tate pairing, we manage to keep the pipeline busy. We also describe the strategies we considered to design our parallel multiplier. They are included in a VHDL code generator allowing for the exploration of a wide range of operators. Then, we outline the architecture of a coprocessor for the Tate pairing over $\mathbb{F}_{3^m}$. However, a final exponentiation is still needed to obtain a unique value, which is desirable in most of the cryptographic protocols. We supplement our pairing accelerator with a coprocessor responsible for this task. An improved exponentiation algorithm allows us to save hardware resources.
According to our place-and-route results on Xilinx FPGAs, our design improves both the computation time and the area-time trade-off compared to previoulsy published coprocessors.

Keywords: Tate pairing, η_T pairing, elliptic curve, finite field arithmetic, Karatsuba-Ofman multiplier, hardware accelerator, FPGA.

1 Introduction

The Weil and Tate pairings were independently introduced in cryptography by Menezes, Okamoto & Vanstone [34] and Frey & Rück [16] as a tool to attack the discrete logarithm problem on some classes of elliptic curves defined over finite fields. The discovery of constructive properties by Mitsunari, Sakai & Kasahara [38], Sakai, Oghishi & Kasahara [42], and Joux [26] initiated the proposal of an ever increasing number of protocols based on bilinear pairings: identity-based

encryption [11], short signature [13], and efficient broadcast encryption [12] to mention but a few.

Miller described the first iterative algorithm to compute the Weil and Tate pairings back in 1986 [35,36]. In practice, the Tate pairing seems to be more efficient for computation (see for instance [20,31]) and has attracted a lot of interest from the research community. Supersingular curves received considerable attention since significant improvements of Miller's algorithm were independently proposed by Barreto *et al.* [4] and Galbraith *et al.* [17] in 2002. One year later, Duursma & Lee gave a closed formula in the case of characteristic three [14]. In 2004, Barreto *et al.* [3] introduced the η_T approach, which further shortens the loop of Miller's algorithm. Recall that the modified Tate pairing can be computed from the reduced η_T pairing at almost no extra cost [7]. More recently, Hess, Smart, and Vercauteren generalized these results to ordinary curves [23, 24, 45].

This paper is devoted to the design of a coprocessor for the Tate pairing on supersingular elliptic curves in characteristic three. We propose here a novel architecture based on a pipelined Karatsuba-Ofman multiplier over $\mathbb{F}_{3^m}$ to implement Miller's algorithm. Thanks to a judicious choice of algorithms for tower field arithmetic and a careful analysis of the scheduling, we manage to keep the pipeline busy and compute one iteration of Miller's algorithm in only 17 clock cycles (Section 2). We describe the strategies we considered to design our parallel multiplier in Section 3. They are included in a VHDL code generator allowing for the exploration of a wide range of operators. Section 4 describes the architecture of a coprocessor for the Tate pairing over $\mathbb{F}_{3^m}$. We summarize our implementation results on FPGA and provide the reader with a thorough comparison against previously published coprocessors in Section 5.

For the sake of concision, we are forced to skip the description of many important concepts of elliptic curve theory. We suggest the interested reader to review [46] for an in-depth coverage of this topic.

2 Reduced η_T Pairing in Characteristic Three Revisited

In the following, we consider the computation of the reduced η_T pairing in characteristic three. Table 1 summarizes the parameters of the algorithm and the supersingular curve. We refer the reader to [3, 8] for more details about the computation of the η_T pairing.

2.1 Computation of Miller's Algorithm

We rewrote the reversed-loop algorithm in characteristic three described in [8], denoting each iteration with parenthesized indices in superscript, in order to emphasize the intrinsic parallelism of the η_T pairing (Algorithm 1). At each iteration of Miller's algorithm, two tasks are performed in parallel, namely: a sparse multiplication over $\mathbb{F}_{3^{6m}}$, and the computation of the coefficients for the next sparse operation. We say that an operand in $\mathbb{F}_{3^{6m}}$ is sparse when some of its coefficients are either zero or one.

Table 1. Supersingular curves over $\mathbb{F}_{3^m}$.

Underlying field	$\mathbb{F}_{3^m}$, where m is coprime to 6
Curve	$E : y^2 = x^3 - x + b$, with $b \in \{-1, 1\}$
Number of rational points	$N = \#E(\mathbb{F}_{3^m}) = 3^m + 1 + \mu b 3^{(m+1)/2}$, with $\mu = \begin{cases} +1 & \text{if } m \equiv 1, 11 \pmod{12}, \text{ or} \\ -1 & \text{if } m \equiv 5, 7 \pmod{12} \end{cases}$
Embedding degree	$k = 6$
Distortion map	$\psi : E(\mathbb{F}_{3^m})[\ell] \longrightarrow E(\mathbb{F}_{3^{6m}})[\ell] \setminus E(\mathbb{F}_{3^m})[\ell]$ $(x, y) \longmapsto (\rho - x, y\sigma)$ with $\rho \in \mathbb{F}_{3^3}$ and $\sigma \in \mathbb{F}_{3^2}$ satisfying $\rho^3 = \rho + b$ and $\sigma^2 = -1$
Tower field	$\mathbb{F}_{3^{6m}} = \mathbb{F}_{3^m}[\rho, \sigma] \cong \mathbb{F}_{3^m}[X, Y]/(X^3 - X - b, Y^2 + 1)$
Final exponentiation	$M = \left(3^{3m} - 1\right) \cdot (3^m + 1) \cdot \left(3^m + 1 - \mu b 3^{(m+1)/2}\right)$
Parameters of Algorithm 1	$\lambda = \begin{cases} +1 & \text{if } m \equiv 7, 11 \pmod{12}, \text{ or} \\ -1 & \text{if } m \equiv 1, 5 \pmod{12}, \text{ and} \end{cases}$ $\nu = \begin{cases} +1 & \text{if } m \equiv 5, 11 \pmod{12}, \text{ or} \\ -1 & \text{if } m \equiv 1, 7 \pmod{12} \end{cases}$

Sparse multiplication over $\mathbb{F}_{3^{6m}}$ (lines 6 and 7). The intermediate result $R^{(i-1)}$ is multiplied by the sparse operand $S^{(i)}$. This operation is easier than a standard multiplication over $\mathbb{F}_{3^{6m}}$.

The choice of a sparse multiplication algorithm over $\mathbb{F}_{3^{6m}}$ requires careful attention. Bertoni *et al.* [6] and Gorla *et al.* [18] took advantage of Karatsuba-Ofman multiplication and Lagrange interpolation, respectively, to reduce the number of multiplications over $\mathbb{F}_{3^m}$ at the expense of several additions (note that Gorla *et al.* study standard multiplication over $\mathbb{F}_{3^{6m}}$ in [18], but extending their approach to sparse multiplication is straightforward). In order to keep the pipeline of a Karatsuba-Ofman multiplier busy, we would have to embed in our processor a large multioperand adder (up to twelve operands for the scheme proposed by Gorla *et al.*) and several multiplexers to deal with the irregular datapath. This would negatively impact the area and the clock frequency, and we prefer considering the algorithm discussed by Beuchat *et al.* in [10] which gives a better trade-off between the number of multiplications and additions over the underlying field when $b = 1$. We give here a more general version of this scheme which also works when $b = -1$ (Algorithm 2). It involves 17 multiplications and 29 additions over $\mathbb{F}_{3^m}$, and reduces the number of intermediate variables compared to the algorithms mentioned above. Another nice feature of this scheme is that it requires the addition of at most four operands.

We suggest to take advantage of a Karatsuba-Ofman multiplier with seven pipeline stages to compute $S^{(i)}$ and $R^{(i-1)} \cdot S^{(i)}$. We managed to find a scheduling that allows us to perform a multiplication over $\mathbb{F}_{3^m}$ at each clock cycle, thus

Algorithm 1 Computation of the reduced η_T pairing in characteristic three. Intermediate variables in uppercase belong to $\mathbb{F}_{3^{6m}}$, those in lowercase to $\mathbb{F}_{3^m}$.

Input: $P = (x_P, y_P)$ and $Q = (x_Q, y_Q) \in E(\mathbb{F}_{3^m})[\ell]$.
Output: $\eta_T(P,Q)^M \in \mathbb{F}^*_{3^{6m}}$.

1: $x_P^{(0)} \leftarrow x_p - \nu b$; $y_p^{(0)} \leftarrow -\mu b y_P$;
2: $x_Q^{(0)} \leftarrow x_Q$; $y_Q^{(0)} \leftarrow -\lambda y_Q$;
3: $t^{(0)} \leftarrow x_P^{(0)} + x_Q^{(0)}$;
4: $R^{(-1)} \leftarrow \lambda y_P^{(0)} \cdot t^{(0)} - \lambda y_Q^{(0)} \sigma - \lambda y_P^{(0)} \rho$;
5: **for** $i = 0$ to $(m-1)/2$ **do**
6: $\quad S^{(i)} \leftarrow -\left(t^{(i)}\right)^2 + y_P^{(i)} y_Q^{(i)} \sigma - t^{(i)} \rho - \rho^2$;
7: $\quad R^{(i)} \leftarrow R^{(i-1)} \cdot S^{(i)}$;
8: $\quad x_P^{(i+1)} \leftarrow \sqrt[3]{x_P^{(i)}}$; $y_P^{(i+1)} \leftarrow \sqrt[3]{y_P^{(i)}}$;
9: $\quad x_Q^{(i+1)} \leftarrow \left(x_Q^{(i)}\right)^3$; $y_Q^{(i+1)} \leftarrow \left(y_Q^{(i)}\right)^3$;
10: $\quad t^{(i+1)} \leftarrow x_P^{(i)} + x_Q^{(i)}$;
11: **end for**
12: **return** $\left(R^{((m-1)/2)}\right)^M$;

keeping the pipeline busy. Therefore, we compute lines 6 and 7 of Algorithm 1 in 17 clock cycles.

Computation of the coefficients of the next sparse operand $S^{(i+1)}$ (lines 8 to 10). The second task consists of computing the sparse operand $S^{(i+1)}$ required for the next iteration of Miller's algorithm. Two cubings and an addition over $\mathbb{F}_{3^m}$ allow us to update the coordinates of point P and to determine the coefficient $t^{(i+1)}$ of the sparse operand $S^{(i+1)}$, respectively.

Recall that the η_T pairing over $\mathbb{F}_{3^m}$ comes in two flavors: the original one involves a cubing over $\mathbb{F}_{3^{6m}}$ after each sparse multiplication. Barreto *et al.* [3] explained how to get rid of that cubing at the price of two cube roots over $\mathbb{F}_{3^m}$ to update the coordinates of point Q. It is essential to consider such an algorithm here in order to minimize the number of arithmetic operations over $\mathbb{F}_{3^m}$ to be performed in the first task (which is the most expensive one).

According to our results, the critical path of the circuit is never located in a cube root operator when pairing-friendly irreducible trinomials or pentanomials [2, 21] are used to define $\mathbb{F}_{3^m}$. If by any chance such polynomials are not available for the considered extension of $\mathbb{F}_3$ and the critical path is in the cube root, it is always possible to pipeline this operation. Therefore, the cost of cube roots is hidden by the first task.

2.2 Final Exponentiation (line 12)

The final step in the computation of the η_T pairing is the final exponentiation, where the result of Miller's algorithm $R^{((m-1)/2)} = \eta_T(P,Q)$ is raised to the

Algorithm 2 Sparse multiplication over $\mathbb{F}_{3^{6m}}$.

Input: $b \in \{-1, 1\}$; $t^{(i)}$, $y_P^{(i)}$, and $y_Q^{(i)} \in \mathbb{F}_{3^m}$; $R^{(i-1)} \in \mathbb{F}_{3^{6m}}$.

Output: $R^{(i)} = R^{(i-1)} \cdot S^{(i)} \in \mathbb{F}_{3^{6m}}$, where $S^{(i)} = \left(-\left(t^{(i)}\right)^2 + y_P^{(i)} y_Q^{(i)} \sigma - t^{(i)} \rho - \rho^2\right)$.

1: $p_0^{(i)} \leftarrow r_0^{(i-1)} \cdot t^{(i)}$; $p_1^{(i)} \leftarrow r_1^{(i-1)} \cdot t^{(i)}$; $p_2^{(i)} \leftarrow r_2^{(i-1)} \cdot t^{(i)}$;
2: $p_3^{(i)} \leftarrow r_3^{(i-1)} \cdot t^{(i)}$; $p_4^{(i)} \leftarrow r_4^{(i-1)} \cdot t^{(i)}$; $p_5^{(i)} \leftarrow r_5^{(i-1)} \cdot t^{(i)}$;
3: $p_6^{(i)} \leftarrow t^{(i)} \cdot t^{(i)}$; $p_7^{(i)} \leftarrow -y_P^{(i)} \cdot y_Q^{(i)}$;
4: $s_0^{(i)} \leftarrow -r_0^{(i-1)} - r_1^{(i-1)}$; $s_1^{(i)} \leftarrow -r_2^{(i-1)} - r_3^{(i-1)}$;
5: $s_2^{(i)} \leftarrow -r_4^{(i-1)} - r_5^{(i-1)}$; $s_3^{(i)} \leftarrow p_6^{(i)} + p_7^{(i)}$;
6: $a_0^{(i)} \leftarrow r_2^{(i-1)} + p_4^{(i)}$; $a_2^{(i)} \leftarrow b r_4^{(i-1)} + p_0^{(i)} + a_0^{(i)}$; $a_4^{(i)} \leftarrow r_0^{(i-1)} + r_4^{(i-1)} + p_2^{(i)}$;
7: $a_1^{(i)} \leftarrow r_3^{(i-1)} + p_5^{(i)}$; $a_3^{(i)} \leftarrow b r_5^{(i-1)} + p_1^{(i)} + a_1^{(i)}$; $a_5^{(i)} \leftarrow r_1^{(i-1)} + r_5^{(i-1)} + p_3^{(i)}$;
8: $p_8^{(i)} \leftarrow r_0^{(i-1)} \cdot p_6^{(i)}$; $p_9^{(i)} \leftarrow r_1^{(i-1)} \cdot p_7^{(i)}$; $p_{10}^{(i)} \leftarrow s_0^{(i)} \cdot s_3^{(i)}$;
9: $p_{11}^{(i)} \leftarrow r_2^{(i-1)} \cdot p_6^{(i)}$; $p_{12}^{(i)} \leftarrow r_3^{(i-1)} \cdot p_7^{(i)}$; $p_{13}^{(i)} \leftarrow s_1^{(i)} \cdot s_3^{(i)}$;
10: $p_{14}^{(i)} \leftarrow r_4^{(i-1)} \cdot p_6^{(i)}$; $p_{15}^{(i)} \leftarrow r_5^{(i-1)} \cdot p_7^{(i)}$; $p_{16}^{(i)} \leftarrow s_2^{(i)} \cdot s_3^{(i)}$;
11: $r_0^{(i)} \leftarrow -b a_0^{(i)} - p_8^{(i)} + p_9^{(i)}$; $r_1^{(i)} \leftarrow -b a_1^{(i)} + p_8^{(i)} + p_9^{(i)} + p_{10}^{(i)}$;
12: $r_2^{(i)} \leftarrow -a_2^{(i)} - p_{11}^{(i)} + p_{12}^{(i)}$; $r_3^{(i)} \leftarrow -a_3^{(i)} + p_{11}^{(i)} + p_{12}^{(i)} + p_{13}^{(i)}$;
13: $r_4^{(i)} \leftarrow -a_4^{(i)} - p_{14}^{(i)} + p_{15}^{(i)}$; $r_5^{(i)} \leftarrow -a_5^{(i)} + p_{14}^{(i)} + p_{15}^{(i)} + p_{16}^{(i)}$;
14: **return** $r_0^{(i)} + r_1^{(i)}\sigma + r_2^{(i)}\rho + r_3^{(i)}\sigma\rho + r_4^{(i)}\rho^2 + r_5^{(i)}\sigma\rho^2$;

M-th power. This exponentiation is necessary since $\eta_T(P, Q)$ is only defined up to N-th powers in $\mathbb{F}^*_{3^{6m}}$.

In order to compute this final exponentiation, we use the algorithm presented by Beuchat *et al.* in [8]. This method exploits the special form of the exponent M (see Table 1) to achieve better performances than with a classical square-and-multiply algorithm. Among other computations, this final exponentiation involves the raising of an element of $\mathbb{F}^*_{3^{6m}}$ to the $3^{(m+1)/2}$-th power, which Beuchat *et al.* [8] perform by computing $(m+1)/2$ successive cubings over $\mathbb{F}^*_{3^{6m}}$. Each of these cubings requiring 6 cubings and 6 additions over $\mathbb{F}_{3^m}$, the total cost of this step is $3m + 3$ cubings and $3m + 3$ additions.

We present here a new method for computing $U^{3^{(m+1)/2}}$ for $U = u_0 + u_1\sigma + u_2\rho + u_3\sigma\rho + u_4\rho^2 + u_5\sigma\rho^2 \in \mathbb{F}^*_{3^{6m}}$ by exploiting the linearity of the Frobenius map (*i.e.* cubing in characteristic three) to reduce the number of additions. Indeed, noting that $\sigma^{3^i} = (-1)^i\sigma$, $\rho^{3^i} = \rho + ib$ and $(\rho^2)^{3^i} = \rho^2 - ib\rho + i^2$, we obtain the following formulae for U^{3^i}, depending on the value of i:

$$U^{3^i} = \left(u_0 - \epsilon_1 u_2 + \epsilon_2 u_4\right)^{3^i} + \epsilon_3 \left(u_1 - \epsilon_1 u_3 + \epsilon_2 u_5\right)^{3^i} \sigma + \left(u_2 + \epsilon_1 u_4\right)^{3^i} \rho + \epsilon_3 \left(u_3 + \epsilon_1 u_5\right)^{3^i} \sigma\rho + u_4^{3^i} \rho^2 + \epsilon_3 u_5^{3^i} \sigma\rho^2,$$

with $\epsilon_1 = -ib \mod 3$, $\epsilon_2 = i^2 \mod 3$, and $\epsilon_3 = (-1)^i$.

Thus, according to the value of $(m + 1)/2$ modulo 6, the computation of $U^{3^{(m+1)/2}}$ will still require $3m + 3$ cubings but at most only 6 additions or subtractions over $\mathbb{F}_{3^m}$.

3 Fully Parallel Karatsuba-Ofman Multipliers Over $\mathbb{F}_{3^m}$

As mentioned in Section 2.1, our hardware accelerator is based on a pipelined Karatsuba-Ofman multiplier over $\mathbb{F}_{3^m}$. This operator is responsible for the computation of the 17 products involved in the sparse multiplication over $\mathbb{F}_{3^{6m}}$ occuring at each iteration of Miller's algorithm. In the following we give a short description of the multiplier block used in this work.

Let $f(x)$ be an irreducible polynomial of degree m over $\mathbb{F}_3$. Then the ternary extension field $\mathbb{F}_{3^m}$ can be defined as $\mathbb{F}_{3^m} \cong \mathbb{F}_3[x]/\left(f(x)\right)$. Multiplication in $\mathbb{F}_{3^m}$ of two arbitrary elements represented as ternary polynomials of degree at most $m-1$ is defined as the polynomial multiplication of the two elements modulo the irreducible polynomial $f(x)$, *i.e.* $c(x) = a(x)b(x) \bmod f(x)$. This implies that we can obtain the field product by first computing the polynomial multiplication of $a(x)$ and $b(x)$ of degree at most $2m-2$ followed by a modular reduction step with $f(x)$. As long as we select $f(x)$ with low Hamming weight (*i.e.* trinomials, tetranomials, *etc.*), the modular reduction step can be accomplished at a linear computational complexity $O(m)$ by using a combination of adders and subtracters over $\mathbb{F}_3$. This implies that the cost of this modular reduction step is much lower than the one associated to polynomial multiplication. In this work, due to its subquadratic space complexity, we used a modified version of the classical Karatsuba-Ofman multiplier for computing the polynomial multiplication step as explained next.

3.1 Variations on the Karatsuba-Ofman Algorithm

The Karatsuba-Ofman multiplier is based on the observation that the polynomial product $c = a \cdot b$ (dropping the (x) notation) can be computed as

$$c = a^L b^L + x^n \left[(a^H + a^L)(b^L + b^H) - (a^H b^H + a^L b^L)\right] + x^{2n} a^H b^H,$$

where $n = \lceil \frac{m}{2} \rceil$, $a = a^L + x^n a^H$, and $b = b^L + x^n b^H$.

Notice that since we are working with polynomials, there is no carry propagation. This allows one to split the operands in a slightly different way: For instance Hanrot and Zimmermann suggested to split them into odd and even part [22]. It was adapted to multiplication over $\mathbb{F}_{2^m}$ by Fan *et al.* [15]. This different way of splitting allows one to save approximatively m additions over $\mathbb{F}_3$ during the reconstruction of the product. This is due to the fact that there is no overlap between the odd and even parts at the reconstruction step, whereas there is some with the higher/lower part splitting method traditionally used.

Another natural way to generalize the Karatsuba-Ofman multiplier is to split the operands not into two, but into three or more parts. That splitting can be done in a classical way, *i.e.* splitting each operand in ascending parts from the lower to the higher powers of x, or splitting them using a generalized odd/even way, *i.e.* according to the degree modulo the number of split parts. By applying this strategy recursively, in each iteration each polynomial multiplication is transformed into three or more polynomial multiplications with their degrees

progressively reduced, until all the polynomial operands collapse into single coefficients. Nevertheless, practice has shown that is better to truncate the recursion earlier, performing the underlying multiplications using alternative techniques that are more compact and/or faster for low-degree operands (typically the so-called school book method with quadratic complexity has been selected).

3.2 A Pipelined Architecture for the Karatsuba-Ofman Multiplier

The field multiplications involved in the reduced η_T pairing do not present dependencies among themselves and thus, it is possible to compute these products using a pipelined architecture. By following this strategy, once that each stage of the pipeline is loaded, we are able to compute one field multiplication over $\mathbb{F}_{3^m}$ every clock cycle. The pipelined architecture was achieved by inserting registers in-between the computation of the partial product operations associated to the *divide-and-conquer* Karatsuba-Ofman strategy, where the depth of the pipeline can be adjusted according to the complexity of the application at hand. This approach allows us to cut the critical path of the whole multiplier structure.

In order to study a wide range of implementation strategies, we decided to write a VHDL code generator tool. This tool allows us to automatically generate the VHDL description of different Karatsuba-Ofman multiplier versions according to several parameters (field extension degree, irreducible polynomial, splitting method, *etc.*). Our automatic tool was extremely useful for selecting the circuit that showed the best time, the smallest area or a good trade-off between them.

4 A Coprocessor for the η_T Pairing in Characteristic Three

As pointed out by Beuchat *et al.* [9], the computation of $R^{((m-1)/2)}$ and the final exponentiation do not share the same datapath and it seems judicious to pipeline these two tasks using two distinct coprocessors in order to reduce the computation time.

4.1 Computation of Miller's Algorithm

A first coprocessor based on a Karatsuba-Ofman multiplier with seven pipeline stages is responsible for computing Miller's loop (Figure 1). We tried to minimize the amount of hardware required to implement the sparse multiplication over $\mathbb{F}_{3^{6m}}$, while keeping the pipeline busy. Besides the parallel multiplier described in Section 3, our architecture consists of four main blocks:

- *Computation of the coefficients of* $S^{(i+1)}$. The first block embeds four registers to store the coordinates of points P and Q. It is responsible for computing $x_P^{(i+1)}$, $y_P^{(i+1)}$, $x_Q^{(i+1)}$, $y_Q^{(i+1)}$, and $t^{(i+1)}$ at each iteration. It also includes dedicated hardware to perform the initialization step of Algorithm 1 (lines 1 and 2).

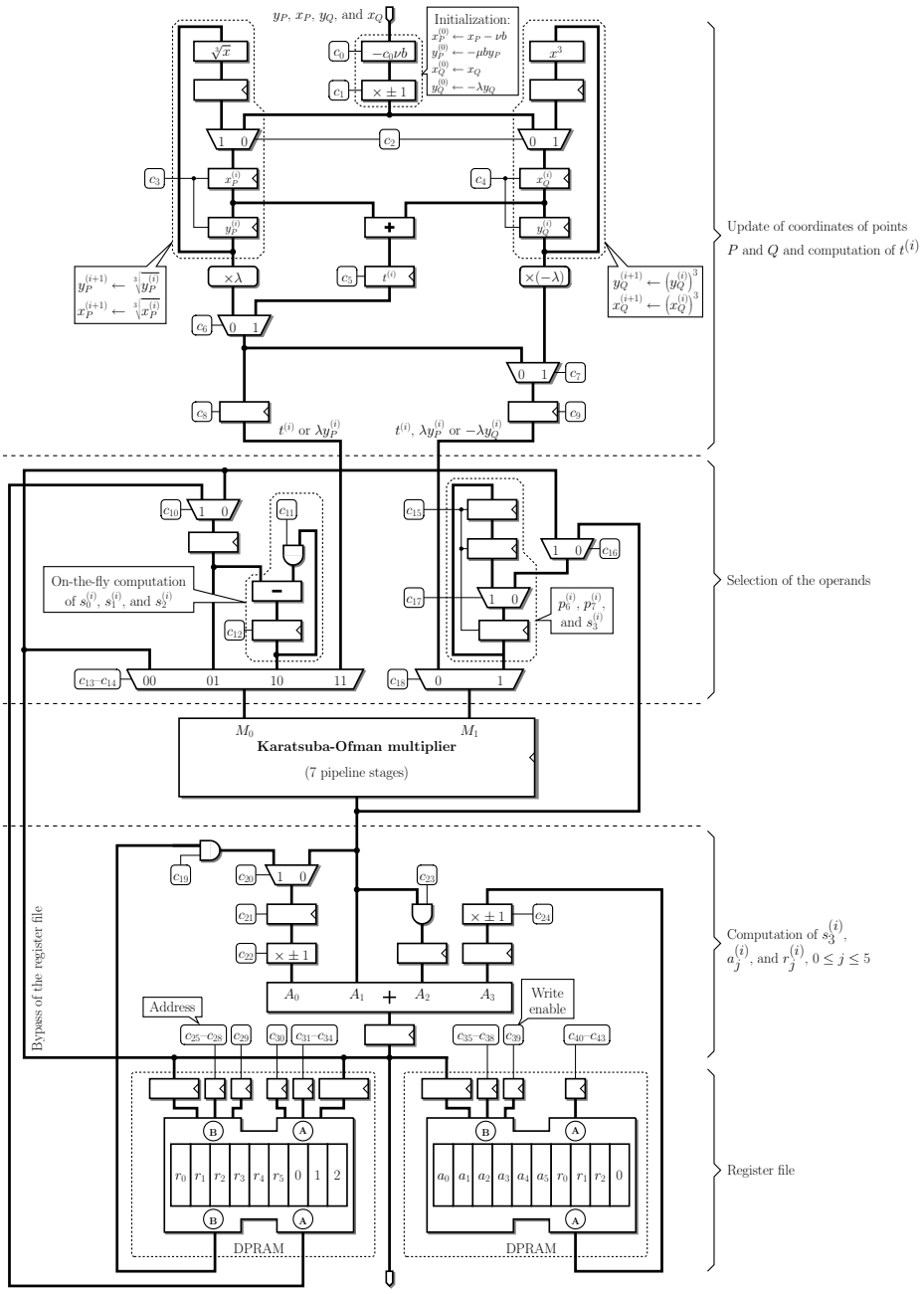

Fig. 1. A coprocessor for the η_T pairing in characteristic three. All control bits c_i belong to $\{0, 1\}$.

- *Selection of the operands of the multiplier.* At each iteration of Miller's algorithm, we have to provide our Karatsuba-Ofman multiplier with $t^{(i)}$, $y_P^{(i)}$, and $y_Q^{(i)}$ in order to compute the coefficients of $S^{(i)}$ (see Algorithm 1, line 6). An accumulator allows us to compute $s_0^{(i)}$, $s_1^{(i)}$, and $s_2^{(i)}$ on-the-fly. We store $p_6^{(i)}$, $p_7^{(i)}$, and $s_3^{(i)}$ in a circular shift register: this approach allows for an easy implementation of lines 8, 9, and 10 of Algorithm 2.
- *Addition over* $\mathbb{F}_{3^m}$. A nice feature of the algorithm we selected for sparse multiplication over $\mathbb{F}_{3^{6m}}$ is that it requires the addition of at most four operands. Thus, it suffices to complement the Karatsuba-Ofman multiplier with a 4-input adder to compute $s_3^{(i)}$, $a_j^{(i)}$, and $r_j^{(i)}$, $0 \leq j \leq 5$. Registers allow us to store several products $p_j^{(i)}$, which is for instance useful when computing $s_3^{(i)} \leftarrow p_6^{(i)} + p_7^{(i)}$.
- *Register file.* The register file is implemented by means of Dual-Ported RAM (DPRAM). In order to avoid memory collisions, we had to split it into two parts and store two copies of $r_0^{(i)}$, $r_1^{(i)}$, and $r_2^{(i)}$.

The initialization step (Algorithm 1, lines 1 to 4) and each iteration of Miller's loop (Algorithm 1, lines 6 to 10) require 17 clock cycles. Therefore, our coprocessor returns $R^{(m-1)/2}$ after $17 \cdot (m+3)/2$ clock cycles.

4.2 Final Exponentiation

Our first attempt at computing the final exponentiation was to use the unified arithmetic operator introduced by Beuchat *et al.* [8]. Unfortunately, due to the sequential scheduling inherent to this operator, it turned out that the final exponentiation algorithm required more clock cycles than the computation of Miller's algorithm by our coprocessor. We therefore had to consider a slightly more parallel architecture.

Noticing that the critical operations in the final exponentiation algorithm were multiplications and long sequences of cubings over $\mathbb{F}_{3^m}$, we designed the coprocessor for arithmetic over $\mathbb{F}_{3^m}$ depicted in Figure 2. Besides a register file implemented by means of DPRAM, our coprocessor embeds a parallel-serial multiplier [44] processing 14 coefficients of an operand at each clock cycle, and a novel unified operator for addition, subtraction, accumulation, Frobenius map (*i.e.* cubing), and double Frobenius map (*i.e.* raising to the 9th power). This architecture allowed us to efficiently implement the final exponentiation algorithm described for instance in [8], while taking advantage of the improvement proposed in Section 2.2.

5 Results and Comparisons

Thanks to our automatic VHDL code generator, we designed several versions of the proposed architecture and prototyped our coprocessors on Xilinx FPGAs with average speedgrade. Table 2 provides the reader with a comparison between

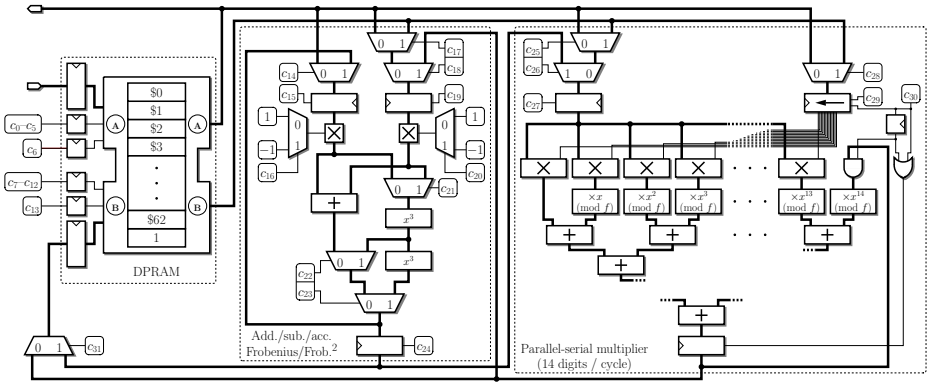

Fig. 2. A coprocessor for the final exponentiation of the η_T pairing in characteristic three.

our work and accelerators for the Tate and the η_T pairing over supersingular (hyper)elliptic curves published in the open literature (our comparison remains fair since the Tate pairing can be computed from the η_T pairing at no extra cost [7]). The third column measures the security of the curve as the key length required by a symmetric-key algorithm of equivalent security. Note that the architecture proposed by Kömürcü & Savas [32] does not implement the final exponentiation, and that Barenghi *et al.* [1] work with a supersingular curve defined over $\mathbb{F}_p$, where p is a 512-bit prime number.

Most of the authors who described hardware accelerators for the Tate pairing over $\mathbb{F}_{3^m}$ considered only low levels of security. Thus, we designed a first architecture for $m = 97$. It simultaneously improves the speed record previously held by Jiang [25], and the Area-Time (AT) product of the coprocessor introduced by Beuchat *et al.* [10].

Then, we studied a more realistic setup and placed-and-routed a second accelerator for $m = 193$, thus achieving a level of security equivalent to 89-bit symmetric encryption. Beuchat *et al.* [7] introduced a unified arithmetic operator in order to reduce the silicon footprint of the circuit to ensure scalability, while trying to minimize the impact on the overall performances. In this work, we focused on the other end of the hardware design spectrum and significantly reduced the computation time reported by Beuchat *et al.* in [7]. A much more unexpected result is that we also improved the AT product. The bottleneck is the usage of the FPGA resources: the unified arithmetic operator allows one to achieve higher levels of security on the same circuit area.

Our architectures are also much faster than software implementations. Mitsunari wrote a very careful multithreaded implementation of the η_T pairing over $\mathbb{F}_{3^{97}}$ and $\mathbb{F}_{3^{193}}$ [37]. He reported a computation time of 92 μs and 553 μs, respectively, on an Intel Core 2 Duo processor (2.66 GHz). Interestingly enough, his software outperforms several hardware architectures proposed by other researchers for low levels of security. When we compare his results with our work,

Table 2. Hardware accelerators for the Tate and η_T pairings.

	Curve	Security [bits]	FPGA	Area [slices]	Freq. [MHz]	Calc. time [μs]	AT prod.
Kerins *et al.* [30]	$E(\mathbb{F}_{3^{97}})$	66	xc2vp125	55616	15	850	47.3
Kömürcü & Savas [32]	$E(\mathbb{F}_{3^{97}})$	66	xc2vp100	14267	77	250.7	3.6
Ronan *et al.* [39]	$E(\mathbb{F}_{3^{97}})$	66	xc2vp100-6	15401	85	183	2.8
Grabher & Page [19]	$E(\mathbb{F}_{3^{97}})$	66	xc2vp4-6	4481	150	432.3	1.9
Jiang [25]	$E(\mathbb{F}_{3^{97}})$	66	xc4vlx200-11	74105	78	20.9	1.55
Beuchat *et al.* [7]	$E(\mathbb{F}_{3^{97}})$	66	xc2vp20-6	4455	105	92	0.4
Beuchat *et al.* [10]	$E(\mathbb{F}_{3^{97}})$	66	xc2vp30-6	10897	147	33	0.36
This work	$E(\mathbb{F}_{3^{97}})$	66	xc2vp30-6	18360	137	6.2	0.11
	$E(\mathbb{F}_{3^{97}})$	66	xc4vlx60-11	18683	179	4.8	0.09
Shu *et al.* [43]	$E(\mathbb{F}_{2^{239}})$	67	xc2vp100-6	25287	84	41	1.04
Beuchat *et al.* [7]	$E(\mathbb{F}_{2^{239}})$	67	xc2vp20-6	4557	123	107	0.49
Keller *et al.* [28]	$E(\mathbb{F}_{2^{251}})$	68	xc2v6000-4	27725	40	2370	65.7
Keller *et al.* [29]	$E(\mathbb{F}_{2^{251}})$	68	xc2v6000-4	13387	40	2600	34.8
Li *et al.* [33]	$E(\mathbb{F}_{2^{283}})$	72	xc4vfx140-11	55844	160	590	32.9
Shu *et al.* [43]	$E(\mathbb{F}_{2^{283}})$	72	xc2vp100-6	37803	72	61	2.3
Ronan *et al.* [40]	$E(\mathbb{F}_{2^{313}})$	75	xc2vp100-6	41078	50	124	5.1
Ronan *et al.* [41]	$C(\mathbb{F}_{2^{103}})$	75	xc2vp100-6	30464	41	132	4.02
Barenghi *et al.* [1]	$E(\mathbb{F}_p)$	87	xc2v8000-5	33857	135	1610	54.5
Beuchat *et al.* [7]	$E(\mathbb{F}_{2^{459}})$	89	xc2vp20-6	8153	115	327	2.66
Beuchat *et al.* [7]	$E(\mathbb{F}_{3^{193}})$	89	xc2vp20-6	8266	90	298	2.46
This work	$E(\mathbb{F}_{3^{193}})$	89	xc2vp125-6	46360	130	12.8	0.59
	$E(\mathbb{F}_{3^{193}})$	89	xc4vlx100-11	47433	167	10.0	0.47

we note that we increase the gap between software and hardware when considering larger values of m. The computation of the Tate pairing over $\mathbb{F}_{3^{193}}$ on a Virtex-4 LX FPGA with a medium speedgrade is for instance roughly fifty times faster than software. This speedup justifies the use of large FPGAs which are now available in servers and supercomputers such as the SGI Altix 4700 platform.

Kammler *et al.* [27] reported the first hardware implementation of the Optimal Ate pairing [45] over a Barreto-Naehrig (BN) curve [5], that is an ordinary curve defined over a prime field $\mathbb{F}_p$ with embedding degree $k = 12$. The proposed design is implemented with a 130 nm standard cell library and computes a pairing in 15.8 ms over a 256-bit BN curve. It is however difficult to make a fair comparison between our respective works: the level of security and the target technology are not the same.

6 Conclusion

We proposed a novel architecture based on a pipelined Karatsuba-Ofman multiplier for the η_T pairing in characteristic three. The main design challenge that we faced was to keep the pipeline continuously busy. Accordingly, we modified the scheduling of the η_T pairing in order to introduce more parallelism in the Miller's

algorithm. Note that our careful re-scheduling should allow one to improve the coprocessor described in [10]. We also introduced a faster way to perform the final exponentiation by taking advantage of the linearity of the cubing operation in characteristic three. Both software and hardware implementations can benefit from this technique.

To our knowledge, the place-and-route results on several Xilinx FPGA devices of our designs improved both the computation time and the area-time trade-off of all the hardware pairing coprocessors previously published in the open literature [1, 7, 10, 19, 25, 28–30, 32, 39–41, 43]. We are also currently applying the same methodology used in this work to design a coprocessor for the Tate pairing over $\mathbb{F}_{2^m}$, with promising preliminary results.

Since the pipeline depth in the Karatsuba-Ofman multiplier is fixed by our scheduling, one could argue that the clock frequency will decrease dramatically for larger values of m. However, at the price of a slightly more complex final exponentiation, we could increase the number of pipeline stages: it suffices to perform the odd and even iterations of the main loop of Algorithm 1 in parallel (we multiply for instance $R^{(2i-2)}$ by $S^{(2i)}$ and $R^{(2i-1)}$ by $S^{(2i+1)}$ in Algorithm 1), so that the multiplier processes two sparse products at the same time. Then, a multiplication over $\mathbb{F}_{3^{6m}}$, performed by the final exponentiation coprocessor, will allow us to compute the $\eta_T(P, Q)$ pairing. We wish to investigate more deeply such architectures in the near future.

Another open problem of our interest is the design of a pairing accelerator providing the level of security of AES-128. Kammler *et al.* [27] proposed a first solution in the case of an ordinary curve. However, many questions remain open: Is it for instance possible to achieve such a level of security in hardware with supersingular (hyper)elliptic curves at a reasonable cost in terms of circuit area? Since several protocols rely on such curves, it seems important to address this problem.

Acknowledgments

The authors would like to thank Nidia Cortez-Duarte and the anonymous referees for their valuable comments. This work was supported by the Japan-France Integrated Action Program (Ayame Junior/Sakura).

The authors would also like to express their deepest gratitude to all the various purveyors of always fresh and lip-smackingly scrumptious raw fish and seafood delicacies from around つくば. Specials thanks go to 蛇の目寿司, 太丸鮨, and やぐら. どうもありがとうございました！

References

1. A. Barenghi, G. Bertoni, L. Breveglieri, and G. Pelosi. A FPGA coprocessor for the cryptographic Tate pairing over $\mathbb{F}_p$. In *Proceedings of the Fourth International Conference on Information Technology: New Generations (ITNG'08)*. IEEE Computer Society, 2008.

2. P.S.L.M. Barreto. A note on efficient computation of cube roots in characteristic 3. Cryptology ePrint Archive, Report 2004/305, 2004.
3. P.S.L.M. Barreto, S.D. Galbraith, C. Ó hÉigeartaigh, and M. Scott. Efficient pairing computation on supersingular Abelian varieties. *Designs, Codes and Cryptography*, 42:239–271, 2007.
4. P.S.L.M. Barreto, H.Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, number 2442 in Lecture Notes in Computer Science, pages 354–368. Springer, 2002.
5. P.S.L.M. Barreto and M. Naehrig. Pairing-friendly elliptic curves of prime order. In B. Preneel and S. Tavares, editors, *Selected Areas in Cryptography – SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2006.
6. G. Bertoni, L. Breveglieri, P. Fragneto, and G. Pelosi. Parallel hardware architectures for the cryptographic Tate pairing. In *Proceedings of the Third International Conference on Information Technology: New Generations (ITNG'06)*. IEEE Computer Society, 2006.
7. J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, and F. Rodríguez-Henríquez. A comparison between hardware accelerators for the modified tate pairing over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$. In S.D. Galbraith and K.G. Paterson, editors, *Pairing-Based Cryptography – Pairing 2008*, number 5209 in Lecture Notes in Computer Science, pages 297–315. Springer, 2008.
8. J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, M. Shirase, and T. Takagi. Algorithms and arithmetic operators for computing the η_T pairing in characteristic three. *IEEE Transactions on Computers*, 57(11):1454–1468, November 2008.
9. J.-L. Beuchat, N. Brisebarre, M. Shirase, T. Takagi, and E. Okamoto. A coprocessor for the final exponentiation of the η_T pairing in characteristic three. In C. Carlet and B. Sunar, editors, *Proceedings of Waifi 2007*, number 4547 in Lecture Notes in Computer Science, pages 25–39. Springer, 2007.
10. J.-L. Beuchat, H. Doi, K. Fujita, A. Inomata, P. Ith, A. Kanaoka, M. Katouno, M. Mambo, E. Okamoto, T. Okamoto, T. Shiga, M. Shirase, R. Soga, T. Takagi, A. Vithanage, and H. Yamamoto. FPGA and ASIC implementations of the η_T pairing in characteristic three. *Computers and Electrical Engineering*, To appear.
11. D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, number 2139 in Lecture Notes in Computer Science, pages 213–229. Springer, 2001.
12. D. Boneh, C. Gentry, and B. Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, number 3621 in Lecture Notes in Computer Science, pages 258–275. Springer, 2005.
13. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, number 2248 in Lecture Notes in Computer Science, pages 514–532. Springer, 2001.
14. I. Duursma and H.S. Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In C.S. Laih, editor, *Advances in Cryptology – ASIACRYPT 2003*, number 2894 in Lecture Notes in Computer Science, pages 111–123. Springer, 2003.
15. H. Fan, J. Sun, M. Gu, and K.-Y. Lam. Overlap-free Karatsuba-Ofman polynomial multiplication algorithm. Cryptology ePrint Archive, Report 2007/393, 2007.
16. G. Frey and H.-G. Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 62(206):865–874, April 1994.

17. S.D. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In C. Fieker and D.R. Kohel, editors, *Algorithmic Number Theory – ANTS V*, number 2369 in Lecture Notes in Computer Science, pages 324–337. Springer, 2002.
18. E. Gorla, C. Puttmann, and J. Shokrollahi. Explicit formulas for efficient multiplication in $\mathbb{F}_{3^{6m}}$. In C. Adams, A. Miri, and M. Wiener, editors, *Selected Areas in Cryptography – SAC 2007*, number 4876 in Lecture Notes in Computer Science, pages 173–183. Springer, 2007.
19. P. Grabher and D. Page. Hardware acceleration of the Tate pairing in characteristic three. In J.R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, number 3659 in Lecture Notes in Computer Science, pages 398–411. Springer, 2005.
20. R. Granger, D. Page, and N.P. Smart. High security pairing-based cryptography revisited. In F. Hess, S. Pauli, and M. Pohst, editors, *Algorithmic Number Theory – ANTS VII*, number 4076 in Lecture Notes in Computer Science, pages 480–494. Springer, 2006.
21. D. Hankerson, A. Menezes, and M. Scott. *Identity-Based Cryptography*, chapter 12 (Software Implementation of Pairings), pages 188–206. Cryptology and Information Security Series. IOS Press, 2009.
22. G. Hanrot and P. Zimmermann. A long note on Mulders' short product. *Journal of Symbolic Computation*, 37(3):391–401, 2004.
23. F. Hess. Pairing lattices. In S.D. Galbraith and K.G. Paterson, editors, *Pairing-Based Cryptography – Pairing 2008*, number 5209 in Lecture Notes in Computer Science, pages 18–38. Springer, 2008.
24. F. Hess, N. Smart, and F. Vercauteren. The Eta pairing revisited. *IEEE Transactions on Information Theory*, 52(10):4595–4602, October 2006.
25. J. Jiang. Bilinear pairing (Eta_T Pairing) IP core. Technical report, City University of Hong Kong – Department of Computer Science, May 2007.
26. A. Joux. A one round protocol for tripartite Diffie-Hellman. In W. Bosma, editor, *Algorithmic Number Theory – ANTS IV*, number 1838 in Lecture Notes in Computer Science, pages 385–394. Springer, 2000.
27. D. Kammler, D. Zhang, P. Schwabe, H. Scharwaechter, M. Langenberg, D. Auras, G. Ascheid, R. Leupers, R. Mathar, and H. Meyr. Designing an ASIP for cryptographic pairings over Barreto-Naehrig curves. Cryptology ePrint Archive, Report 2009/056, 2009.
28. M. Keller, T. Kerins, F. Crowe, and W.P. Marnane. FPGA implementation of a GF(2^m) Tate pairing architecture. In K. Bertels, J.M.P. Cardoso, and S. Vassiliadis, editors, *International Workshop on Applied Reconfigurable Computing (ARC 2006)*, number 3985 in Lecture Notes in Computer Science, pages 358–369. Springer, 2006.
29. M. Keller, R. Ronan, W.P. Marnane, and C. Murphy. Hardware architectures for the Tate pairing over GF(2^m). *Computers and Electrical Engineering*, 33(5–6):392–406, 2007.
30. T. Kerins, W.P. Marnane, E.M. Popovici, and P.S.L.M. Barreto. Efficient hardware for the Tate pairing calculation in characteristic three. In J.R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, number 3659 in Lecture Notes in Computer Science, pages 412–426. Springer, 2005.
31. N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. In N.P. Smart, editor, *Cryptography and Coding*, number 3796 in Lecture Notes in Computer Science, pages 13–36. Springer, 2005.

32. G. Kömürcü and E. Savaş. An efficient hardware implementation of the Tate pairing in characteristic three. In E. Prasolova-Førland and M. Popescu, editors, *Proceedings of the Third International Conference on Systems – ICONS 2008*, pages 23–28. IEEE Computer Society, 2008.
33. H. Li, J. Huang, P. Sweany, and D. Huang. FPGA implementations of elliptic curve cryptography and Tate pairing over a binary field. *Journal of Systems Architecture*, 54:1077–1088, 2008.
34. A. Menezes, T. Okamoto, and S.A. Vanstone. Reducing elliptic curves logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, September 1993.
35. V.S. Miller. Short programs for functions on curves. Available at http://crypto.stanford.edu/miller, 1986.
36. V.S. Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, 2004.
37. S. Mitsunari. A fast implementation of η_T pairing in characteristic three on Intel Core 2 Duo processor. Cryptology ePrint Archive, Report 2009/032, 2009.
38. S. Mitsunari, R. Sakai, and M. Kasahara. A new traitor tracing. *IEICE Trans. Fundamentals*, E85–A(2):481–484, Feb 2002.
39. R. Ronan, C. Murphy, T. Kerins, C. Ó hÉigeartaigh, and P.S.L.M. Barreto. A flexible processor for the characteristic 3 η_T pairing. *Int. J. High Performance Systems Architecture*, 1(2):79–88, 2007.
40. R. Ronan, C. Ó hÉigeartaigh, C. Murphy, M. Scott, and T. Kerins. FPGA acceleration of the Tate pairing in characteristic 2. In *Proceedings of the IEEE International Conference on Field Programmable Technology – FPT 2006*, pages 213–220. IEEE, 2006.
41. R. Ronan, C. Ó hÉigeartaigh, C. Murphy, M. Scott, and T. Kerins. Hardware acceleration of the Tate pairing on a genus 2 hyperelliptic curve. *Journal of Systems Architecture*, 53:85–98, 2007.
42. R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. In *2000 Symposium on Cryptography and Information Security (SCIS2000), Okinawa, Japan*, pages 26–28, January 2000.
43. C. Shu, S. Kwon, and K. Gaj. FPGA accelerated Tate pairing based cryptosystem over binary fields. In *Proceedings of the IEEE International Conference on Field Programmable Technology – FPT 2006*, pages 173–180. IEEE, 2006.
44. L. Song and K.K. Parhi. Low energy digit-serial/parallel finite field multipliers. *Journal of VLSI Signal Processing*, 19(2):149–166, July 1998.
45. F. Vercauteren. Optimal pairings. Cryptology ePrint Archive, Report 2008/096, 2008.
46. L.C. Washington. *Elliptic Curves – Number Theory and Cryptography*. CRC Press, 2nd edition, 2008.